# Package ‘conf’

May 20, 2022

**Type** Package

**Title** Visualization and Analysis of Statistical Measures of Confidence

**Version** 1.7.1

**Maintainer** Christopher Weld <ceweld@email.wm.edu>

**Imports** graphics, stats, statmod, fitdistrplus, pracma, rootSolve, utils

**Description** Enables: (1) plotting two-dimensional confidence regions, (2) coverage analysis of confidence region simulations and (3) calculating confidence intervals and the associated actual coverage for binomial proportions. Each is given in greater detail next.
(1) Plots the two-dimensional confidence region for probability distribution parameters (supported distribution suffixes: cauchy, gamma, invgauss, logis, llogis, lnorm, norm, unif, weibull) corresponding to a user-given complete or right-censored dataset and level of significance. The crplot() algorithm plots more points in areas of greater curvature to ensure a smooth appearance throughout the confidence region boundary. An alternative heuristic plots a specified number of points at roughly uniform intervals along its boundary. Both heuristics build upon the radial profile log-likelihood ratio technique for plotting confidence regions given by Jaeger (2016) <doi:10.1080/00031305.2016.1182946>, and are detailed in a publication by Weld (2019) <doi:10.1080/00031305.2018.1564696>.
(2) Performs confidence region coverage simulations for a random sample drawn from a user-specified parametric population distribution, or for a user-specified dataset and point of interest with coversim(). (3) Calculates confidence interval bounds for a binomial proportion with binomTest(), calculates the actual coverage with binomTestCoverage(), and plots the actual coverage with binomTestCoveragePlot(). Calculates confidence interval bounds for the binomial proportion using an ensemble of constituent confidence intervals with binomTestEnsemble(). Calculates confidence interval bounds for the binomial proportion using a complete enumeration of all possible transitions from one actual coverage acceptance curve to another which minimizes the root mean square error for n <= 15 and follows the transitions for well-known confidence intervals for n > 15 using binomTestMSE().

**Depends** R (>= 3.2.0)

**License** GPL (<= 2)

**Encoding** UTF-8

**RoxygenNote** 7.2.0

**Suggests** knitr, rmarkdown

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Christopher Weld [aut, cre] (<https://orcid.org/0000-0001-5902-9738>),
Hayeon Park [aut],
Kexin Feng [aut],
Heather Sasinowska [aut],
Lawrence Leemis [aut],
Andrew Loh [ctb],
Yuan Chang [ctb],
Brock Crook [ctb],
Xin Zhang [ctb]

**Repository** CRAN

**Date/Publication** 2022-05-20 18:30:02 UTC

## R topics documented:

---

`binomTest` *Confidence Intervals for Binomial Proportions*

---

### Description

Generates lower and upper confidence interval limits for a binomial proportion using different types of confidence intervals.

### Usage

```
binomTest(n, x,
          alpha = 0.05,
          intervalType = "Clopper-Pearson")
```

## Arguments

| | |
|---|---|
| `n` | sample size |
| `x` | number of successes |
| `alpha` | significance level for confidence interval |
| `intervalType` | type of confidence interval used; either "Clopper-Pearson", "Wald", "Wilson-Score", "Jeffreys", "Agresti-Coull", "Arcsine", or "Blaker" |

## Details

Generates a lower and upper confidence interval limit for a binomial proportion using

- various types of confidence intervals,
- various sample sizes, and
- various numbers of successes.

When the `binomTest` function is called, it returns a two-element vector in which

- the first element is the lower bound of the confidence interval, and
- the second element is the upper bound of the confidence interval.

This confidence interval is constructed by calculating lower and upper bounds associated with the confidence interval procedure specified by the `intervalType` argument. Lower bounds that are negative are set to 0 and upper bounds that are greater than 1 are set to 1.

## Author(s)

Hayeon Park (<`hpark03@email.wm.edu`>), Larry Leemis (<`leemis@math.wm.edu`>)

## See Also

`dbinom`

## Examples

```
binomTest(10, 6)
binomTest(100, 30, intervalType = "Agresti-Coull")
```

## binomTestCoverage *Actual Coverage Calculation for Binomial Proportions*

### Description

Calculates the actual coverage of a confidence interval for a binomial proportion for a particular sample size $n$ and a particular value of the probability of success $p$ for several confidence interval procedures.

### Usage

```
binomTestCoverage(n, p,
                  alpha = 0.05,
                  intervalType = "Clopper-Pearson")
```

### Arguments

| | |
|---|---|
| `n` | sample size |
| `p` | population probability of success |
| `alpha` | significance level for confidence interval |
| `intervalType` | type of confidence interval used; either "Clopper-Pearson", "Wald", "Wilson-Score", "Jeffreys", "Agresti-Coull", "Arcsine", or "Blaker" |

### Details

Calculates the actual coverage of a confidence interval procedure at a particular value of $p$ for

- various types of confidence intervals,
- various probabilities of success $p$, and
- various sample sizes $n$.

The actual coverage for a particular value of $p$, the probability of success of interest, is

$$c(p) = \sum_{x=0}^{n} I(x,p) \binom{n}{x} p^x (1-p)^{n-x},$$

where $I(x,p)$ is an indicator function that determines whether a confidence interval covers $p$ when $X = x$ (see Vollset, 1993).

The binomial distribution with arguments `size` = $n$ and `prob` = $p$ has probability mass function

$$p(x) = \binom{n}{x} p^x (1-p)^{n-x}$$

for $x = 0, 1, 2, \ldots, n$.

The algorithm for computing the actual coverage for a particular probability of success begins by calculating all possible lower and upper bounds associated with the confidence interval procedure specified by the `intervalType` argument. The appropriate binomial probabilities are summed to determine the actual coverage at $p$.

### Author(s)

Hayeon Park (<hpark03@email.wm.edu>), Larry Leemis (<leemis@math.wm.edu>)

### References

Vollset, S.E. (1993). Confidence Intervals for a Binomial Proportion. Statistics in Medicine, 12, 809-824.

### See Also

dbinom

### Examples

```
binomTestCoverage(6, 0.4)
binomTestCoverage(n = 10, p = 0.3, alpha = 0.01, intervalType = "Wilson-Score")
```

---

## binomTestCoveragePlot *Coverage Plots for Binomial Proportions*

---

### Description

Generates plots for the actual coverage of a binomial proportion using various types of confidence intervals. Plots the actual coverage for a given sample size and stated nominal coverage 1− alpha.

### Usage

```
binomTestCoveragePlot(n,
                      alpha = 0.05,
                      intervalType = "Clopper-Pearson",
                      plo = 0,
                      phi = 1,
                      clo = 1 - 2 * alpha,
                      chi = 1,
                      points = 5 + floor(250 / n),
                      showTrueCoverage = TRUE,
                      gridCurves = FALSE)
```

### Arguments

| | |
|---|---|
| n | sample size |
| alpha | significance level for confidence interval |
| intervalType | type of confidence interval used; either "Clopper-Pearson", "Wald", "Wilson-Score", "Jeffreys", "Agresti-Coull", "Arcsine", or "Blaker" |
| plo | lower limit for percentile (horizontal axis) |
| phi | upper limit for percentile (horizontal axis) |

| | |
|---|---|
| `clo` | lower limit for coverage (vertical axis) |
| `chi` | upper limit for coverage (vertical axis) |
| `points` | number of points plotted in each segment of the plot; if default, varies with 'n' (see above) |
| `showTrueCoverage` | logical; if `TRUE` (default), a solid red line will appear at $1-$ `alpha` |
| `gridCurves` | logical; if `TRUE`, display acceptance curves in gray |

## Details

Generates an actual coverage plot for binomial proportions using

- various types of confidence intervals, and
- various sample sizes.

When the function is called with default arguments,

- the horizontal axis is the percentile at which the coverage is evaluated,
- the vertical axis is the actual coverage percentage at each percentile, that is, the probability that the true value at a percentile is contained in the corresponding confidence interval, and
- the solid red line is the stated coverage of $1-$ `alpha`.

The actual coverage for a particular value of $p$, the percentile of interest, is

$$c(p) = \sum_{x=0}^{n} I(x,p) \binom{n}{x} p^x (1-p)^{n-x},$$

where $I(x,p)$ is an indicator function that determines whether a confidence interval covers $p$ when $X = x$ (see Vollset, 1993).

The binomial distribution with arguments `size` = $n$ and `prob` = $p$ has probability mass function

$$p(x) = \binom{n}{x} p^x (1-p)^{n-x}$$

for $x = 0, 1, \ldots, n$.

The algorithm for plotting the actual coverage begins by calculating all possible lower and upper bounds associated with the confidence interval procedure specified by the `intervalType` argument. These values are concatenated into a vector which is sorted. Negative values and values that exceed 1 are removed from this vector. These values are the breakpoints in the actual coverage function. The `points` argument gives the number of points plotted on each segment of the graph of the actual coverage.

The `plo` and `phi` arguments can be used to expand or compress the plots horizontally.

The `clo` and `chi` arguments can be used to expand or compress the plots vertically.

By default, the `showTrueCoverage` argument plots a solid horizontal red line at the height of the stated coverage. The actual coverage is plotted with solid black lines for each segment of the actual coverage.

The `gridCurves` argument is assigned a logical value which indicates whether the acceptance curves giving all possible actual coverage values should be displayed as gray curves.

## Author(s)

Hayeon Park (<hpark03@email.wm.edu>), Larry Leemis (<leemis@math.wm.edu>)

## References

Vollset, S.E. (1993). Confidence Intervals for a Binomial Proportion. Statistics in Medicine, 12, 809–824.

## See Also

dbinom

## Examples

```
binomTestCoveragePlot(6)
binomTestCoveragePlot(10, intervalType = "Wilson-Score", clo = 0.8)
binomTestCoveragePlot(n = 100, intervalType = "Wald", clo = 0, chi = 1, points = 30)
```

---

## binomTestEnsemble *Ensemble Confidence Intervals for Binomial Proportions*

---

## Description

Generates lower and upper confidence interval limits for a binomial proportion using an ensemble of confidence intervals.

## Usage

```
binomTestEnsemble(n, x,
                  alpha = 0.05,
                  CP = TRUE,
                  WS = TRUE,
                  JF = TRUE,
                  AC = TRUE,
                  AR = TRUE)
```

## Arguments

| | |
|---|---|
| n | sample size |
| x | number of successes |
| alpha | significance level for confidence interval |
| CP | logical; if TRUE (default), include Clopper-Pearson confidence interval procedure in the ensemble |
| WS | logical; if TRUE (default), include Wilson-Score confidence interval procedure in the ensemble |

| | |
|---|---|
| `JF` | logical; if `TRUE` (default), include Jeffreys confidence interval procedure in the ensemble |
| `AC` | logical; if `TRUE` (default), include Agresti-Coull confidence interval procedure in the ensemble |
| `AR` | logical; if `TRUE` (default), include Arcsine confidence interval procedure in the ensemble |

## Details

Generates lower and upper confidence interval limits for a binomial proportions using

- various sample sizes,
- various numbers of successes, and
- various combinations of confidence intervals.

When the `binomTestEnsemble` function is called, it returns a two-element vector in which

- the first element is the lower bound of the Ensemble confidence interval, and
- the second element is the upper bound of the Ensemble confidence interval.

To construct an Ensemble confidence interval that attains an actual coverage that is close to the stated coverage, the five constituent confidence interval procedures can be combined. Since these intervals vary in width, the lower limits and the actual coverage of the constituent confidence intervals at the maximum likelihood estimator are calculated. Likewise, the upper limits and the actual coverage of the constituent confidence intervals at the maximum likelihood estimator are calculated. The centroids of the lower and upper constituent confidence intervals for points falling below and above the stated coverage are connected with a line segment. The point of intersection of these line segments and the stated coverage gives the lower and upper bound of the Ensemble confidence interval. Special cases to this approach are given in the case of (a) the actual coverages all fall above or below the stated coverage, and (b) the slope of the line connecting the centroids is infinite.

If only one of the logical arguments is `TRUE`, the code returns a simple confidence interval of that one procedure.

The Wald confidence interval is omitted because it degenerates in actual coverage for $x = 0$ and $x = n$.

## Author(s)

Hayeon Park (<`hpark03@email.wm.edu`>), Larry Leemis (<`leemis@math.wm.edu`>)

## Examples

```
binomTestEnsemble(10, 3)
binomTestEnsemble(100, 82, CP = FALSE, AR = FALSE)
binomTestEnsemble(33, 1, CP = FALSE, JF = FALSE, AC = FALSE, AR = FALSE)
```

---

`binomTestMSE` *RMSE-Minimizing Confidence Intervals for Binomial Proportions*

---

## Description

Generates lower and upper confidence interval limits for a binomial proportion that minimizes the root mean square error (RMSE) of the actual coverage function.

## Usage

```
binomTestMSE(n, x,
             alpha = 0.05,
             smooth = 1,
             showRMSE = TRUE,
             showAll = FALSE)
```

## Arguments

| | |
|---|---|
| `n` | sample size |
| `x` | number of successes |
| `alpha` | significance level for confidence interval |
| `smooth` | smoothness index |
| `showRMSE` | a logical variable indicating whether to show the value of RSME |
| `showAll` | a logical variable indicating whether to show confidence intervals of all possible number of successes |

## Details

Generates lower and upper confidence interval limits for a binomial proportion for

- various sample sizes,
- various numbers of successes.

When the `binomTestMSE` function is called, it returns a two-element vector in which

- the first element is the lower bound of the RMSE-minimizing confidence interval, and
- the second element is the upper bound of the RMSE-minimizing confidence interval.

An RMSE-minimizing two-sided 100 * (1 - alpha) percent confidence interval for p is constructed from a random sample of size n from a Bernoulli(p) population. The parameter `x` gives the number of successes in the n mutually independent Bernoulli trials. For n <= 15, all possible jumps between acceptance curves associated with the actual coverage function are enumerated based on their one-to-one relationship with the symmetric Dyck paths. For each sequence of jumps between acceptance curves, the confidence interval bounds that are returned are associated with discontinuities in the actual coverage function that together result in the lowest possible RMSE. A set of smoothness constraints that build on four existing non-conservative confidence intervals (Wilson-score, Jeffreys, Arcsine, and Agresti-Coull) is used if the smoothness index `smooth` is set to one. These constraints ensure that the RMSE-confidence interval achieves smoothness, a preferable property of the binomial confidence interval that is related to lower bound differences for adjacent values of `x`. There is a trade-off between the RMSE and the smoothness. For n > 100, smoothness is required. The RMSE usually increases if the smoothness constraints are used. For n > 15, only the symmetric Dyck paths associated with the Wilson–score, Jeffreys, Arcsine, and Agresti–Coull confidence interval procedures are used instead of enumerating because the computation time increases in a factorial fashion in n. The minimal RMSE is not guaranteed for n > 15 because another symmetric Dyck path other than those associated with the four existing confidence interval procedures might prove to be optimal. However, this procedure does ensure a lower RMSE than any of the four existing confidence intervals for all n.

## Author(s)

Kexin Feng (<`kfeng01@email.wm.edu`>), Larry Leemis (<`leemis@math.wm.edu`>), Heather Sasinowska (<`hdsasinowska@wm.edu`>)

## Examples

```
binomTestMSE(10, 3)
```

---

`conf` *conf: Visualization and Analysis of Statistical Measures of Confidence*

---

## Description

Enables:

1. confidence region plots in two-dimensions corresponding to a user given dataset, level of significance, and parametric probability distribution (supported distribution suffixes: cauchy, gamma, invgauss, lnorm, llogis, logis, norm, unif, weibull),
2. coverage simulations (if a point of interest is within or outside of a confidence region boundary) for either random samples drawn from a user-specified parametric distribution or for a user-specified dataset and point of interest, and
3. calculating confidence intervals and the associated actual coverage for binomial proportions.

**Request from authors**: Please properly cite any use of this package and/or its algorithms, which are detailed in the corresponding publication by Weld (2018) <doi:10.1080/00031305.2018.1564696>. Additionally, we welcome and appreciate your feedback and insights as to how this resource is being leveraged to improve whatever it is you do. Please include your name and academic and/or business affiliation in your correspondance.

## Details

This package includes the functions:

- confidence region plots: `crplot`,
- confidence region coverage analysis: `coversim`,

- confidence intervals for binomial proportions: `binomTest`,
- actual coverage calculation for binomial proportions: `binomTestCoverage`,
- actual coverage plots for binomial proportions: `binomTestCoveragePlot`, and
- ensemble confidence intervals for binomial proportions: `binomTestEnsemble`.

## Vignettes

The CRAN website https://CRAN.R-project.org/package=conf contains links for vignettes on the `crplot` and `coversim` functions.

## Acknowledgments

The lead author thanks The Omar Bradley Fellowship for Research in Mathematics for funding that partially supported this work.

## Author(s)

Christopher Weld, Hayeon Park, Larry Leemis

Maintainer: Christopher Weld <ceweld@email.wm.edu>

---

`coversim` *Confidence Region Coverage*

---

## Description

Creates a confidence region and determines coverage results for a corresponding point of interest. Iterates through a user specified number of trials. Each trial uses a random dataset with user-specified parameters (default) or a user specified dataset matrix (`'n'` samples per column, `'iter'` columns) and returns the corresponding actual coverage results. See the CRAN website https://CRAN.R-project.org/package=conf for a link to a `coversim` vignette.

## Usage

```
coversim(alpha, distn,
                n          = NULL,
                iter       = NULL,
                dataset    = NULL,
                point      = NULL,
                seed       = NULL,
                a          = NULL,
                b          = NULL,
                kappa      = NULL,
                lambda     = NULL,
                mu         = NULL,
                s          = NULL,
                sigma      = NULL,
                theta      = NULL,
                heuristic  = 1,
                maxdeg     = 5,
                ellipse_n  = 4,
                pts        = FALSE,
                mlelab     = TRUE,
                sf         = c(5, 5),
                mar        = c(4, 4.5, 2, 1.5),
                xlab       = "",
                ylab       = "",
                main       = "",
                xlas       = 0,
                ylas       = 0,
                origin     = FALSE,
                xlim       = NULL,
                ylim       = NULL,
                tol        = .Machine$double.eps ^ 1,
                info       = FALSE,
                returnsamp  = FALSE,
                returnquant = FALSE,
                repair     = TRUE,
                exact      = FALSE,
                showplot   = FALSE,
                delay      = 0 )
```

**Arguments**

| | |
|---|---|
| `alpha` | significance level; scalar or vector; resulting plot illustrates a 100(1 - `alpha`)% confidence region. |
| `distn` | distribution to fit the dataset to; accepted values: `'cauchy'`, `'gamma'`, `'invgauss'`, `'logis'`, `'llogis'`, `'lnorm'`, `'norm'`, `'unif'`, `'weibull'`. |
| `n` | trial sample size (producing each confidence region); scalar or vector; needed if a dataset is not given. |
| `iter` | iterations (or replications) of individual trials per parameterization; needed if a dataset is not given. |
| `dataset` | a `'n'` x `'iter'` matrix of dataset values, or a vector of length `'n'` (for a single iteration). |
| `point` | coverage is assessed relative to this point. |
| `seed` | random number generator seed. |
| `a` | distribution parameter (when applicable). |
| `b` | distribution parameter (when applicable). |
| `kappa` | distribution parameter (when applicable). |
| `lambda` | distribution parameter (when applicable). |
| `mu` | distribution parameter (when applicable). |
| `s` | distribution parameter (when applicable). |

| | |
|---|---|
| `sigma` | distribution parameter (when applicable). |
| `theta` | distribution parameter (when applicable). |
| `heuristic` | numeric value selecting method for plotting: 0 for elliptic-oriented point distribution, and 1 for smoothing boundary search heuristic. |
| `maxdeg` | maximum angle tolerance between consecutive plot segments in degrees. |
| `ellipse_n` | number of roughly equidistant confidence region points to plot using the elliptic-oriented point distribution (must be a multiple of four because its algorithm exploits symmetry in the quadrants of an ellipse). |
| `pts` | displays confidence region boundary points if `TRUE` (applies to confidence region plots in which `showplot = TRUE`). |
| `mlelab` | logical argument to include the maximum likelihood estimate coordinate point (default is `TRUE`, applies to confidence region plots when `showplot = TRUE`). |
| `sf` | significant figures in axes labels specified using sf = c(x, y), where x and y represent the optional digits argument in the R function `round` as it pertains the horizontal and vertical labels. |
| `mar` | specifies margin values for `par(mar = c( ))` (see `mar` in `par`). |
| `xlab` | string specifying the horizontal axis label (applies to confidence region plots when `showplot = TRUE`). |
| `ylab` | string specifying the vertical axis label (applies to confidence region plots when `showplot = TRUE`). |
| `main` | string specifying the plot title (applies to confidence region plots when `showplot = TRUE`). |
| `xlas` | numeric in 0, 1, 2, 3 specifying the style of axis labels (see `las` in `par`, applies to confidence region plots when `showplot = TRUE`). |
| `ylas` | numeric in 0, 1, 2, 3 specifying the style of axis labels (see `las` in `par`, applies to confidence region plots when `showplot = TRUE`). |
| `origin` | logical argument to include the plot origin (applies to confidence region plots when `showplot = TRUE`). |
| `xlim` | two element vector containing horizontal axis minimum and maximum values (applies to confidence region plots when `showplot = TRUE`). |
| `ylim` | two element vector containing vertical axis minimum and maximum values (applies to confidence region plots when `showplot = TRUE`). |
| `tol` | the `uniroot` parameter specifying its required accuracy. |
| `info` | logical argument to return coverage information in a list; includes `alpha` value(s), `n` value(s), coverage and error results per iteration, and `returnsamp` and/or `returnquant` when requested. |
| `returnsamp` | logical argument; if `TRUE` returns random samples used in a matrix with `n` rows, `iter` cols. |
| `returnquant` | logical argument; if `TRUE` returns random quantiles used in a matrix with `n` rows, `iter` cols. |
| `repair` | logical argument to repair regions inaccessible using a radial angle from its MLE (multiple root azimuths). |

| | |
|---|---|
| `exact` | logical argument specifying if alpha value is adjusted to compensate for negative coverage bias in order to achieve (1 - alpha) coverage probability using previously recorded Monte Carlo simulation results; available for limited values of alpha (roughly <= 0.2–0.3), n (typically n = 4, 5, ..., 50) and distributions (distn suffixes: weibull, llogis, norm). |
| `showplot` | logical argument specifying if each coverage trial produces a plot. |
| `delay` | numeric value of delay (in seconds) between trials so its plot can be seen (applies when `showplot = TRUE`). |

## Details

Parameterizations for supported distributions are given following the default axes convention in use by `crplot` and `coversim`, which are:

| | Horizontal | Vertical |
|---|---|---|
| Distribution | Axis | Axis |
| Cauchy | $a$ | $s$ |
| gamma | $\theta$ | $\kappa$ |
| inverse Gaussian | $\mu$ | $\lambda$ |
| log logistic | $\lambda$ | $\kappa$ |
| log normal | $\mu$ | $\sigma$ |
| logistic | $\mu$ | $\sigma$ |
| normal | $\mu$ | $\sigma$ |
| uniform | $a$ | $b$ |
| Weibull | $\kappa$ | $\lambda$ |

Each respective distribution is defined below.

- The Cauchy distribution for the real-numbered location parameter $a$, scale parameter $s$, and $x$ is a real number, has the probability density function
$$1/(s\pi(1 + ((x - a)/s)^2)).$$
- The gamma distribution for shape parameter $\kappa > 0$, scale parameter $\theta > 0$, and $x > 0$, has the probability density function
$$1/(Gamma(\kappa)\theta^\kappa)x^{(\kappa-1)}exp(-x/\theta).$$
- The inverse Gaussian distribution for mean $\mu > 0$, shape parameter $\lambda > 0$, and $x > 0$, has the probability density function
$$\sqrt(\lambda/(2\pi x^3))exp(-\lambda(x - \mu)^2/(2\mu^2 x)).$$
- The log logistic distribution for scale parameter $\lambda > 0$, shape parameter $\kappa > 0$, and $x \geq 0$, has a probability density function
$$(\kappa\lambda)(x\lambda)^{(\kappa-1)}/(1 + (\lambda x)^\kappa)^2.$$
- The log normal distribution for the real-numbered mean $\mu$ of the logarithm, standard deviation $\sigma > 0$ of the logarithm, and $x > 0$, has the probability density function
$$1/(x\sigma\sqrt(2\pi))exp(-(\log x - \mu)^2/(2\sigma^2)).$$

- The logistic distribution for the real-numbered location parameter $\mu$, scale parameter $\sigma$, and $x$ is a real number, has the probability density function
$$(1/\sigma)exp((x-\mu)/\sigma)(1+exp((x-\mu)/\sigma))^{-2}$$
- The normal distribution for the real-numbered mean $\mu$, standard deviation $\sigma > 0$, and $x$ is a real number, has the probability density function
$$1/\sqrt(2\pi\sigma^2)exp(-(x-\mu)^2/(2\sigma^2)).$$
- The uniform distribution for real-valued parameters $a$ and $b$ where $a < b$ and $a \leq x \leq b$, has the probability density function
$$1/(b-a).$$
- The Weibull distribution for scale parameter $\lambda > 0$, shape parameter $\kappa > 0$, and $x > 0$, has the probability density function
$$\kappa(\lambda^\kappa)x^{(\kappa-1)}exp(-(\lambda x)^\kappa).$$

## Value

If the optional argument `info = TRUE` is included then a list of coverage results is returned. That list includes `alpha` value(s), `n` value(s), coverage and error results per iteration. Additionally, `returnsamp = TRUE` and/or `returnquant = TRUE` will result in an `n` row, `iter` column maxtix of sample and/or sample cdf values.

## Author(s)

Christopher Weld (<`ceweld@email.wm.edu`>)

Lawrence Leemis (<`leemis@math.wm.edu`>)

## References

Weld, C., Loh, A., Leemis, L. (in press), "Plotting Likelihood-Ratio Based Confidence Regions for Two-Parameter Univariate Probability Models", The American Statistician.

## See Also

`crplot`, `uniroot`

## Examples

```
## assess actual coverage at various alpha = {0.5, 0.1} given n = 30 samples,  completing
## 10 trials per parameterization (iter) for a normal(mean = 2, sd = 3) rv
coversim(alpha = c(0.5, 0.1), "norm", n = 30, iter = 10, mu = 2, sigma = 3)

## show plots for 5 iterations of 30 samples each from a Weibull(2, 3)
coversim(0.5, "weibull", n = 30, iter = 5, lambda = 1.5, kappa = 0.5, showplot = TRUE,
origin = TRUE)
```

## crplot *Plotting Two-Dimensional Confidence Regions*

### Description

Plots the two-dimensional confidence region for probability distribution parameters (supported distribution suffixes: cauchy, gamma, invgauss, lnorm, llogis, logis, norm, unif, weibull) corresponding to a user given complete or right-censored dataset and level of significance. See the CRAN website https://CRAN.R-project.org/package=conf for a link to two `crplot` vignettes.

### Usage

```
crplot(dataset, alpha, distn,
                cen       = rep(1, length(dataset)),
                heuristic = 1,
                maxdeg    = 5,
                ellipse_n = 4,
                pts       = TRUE,
                mlelab    = TRUE,
                sf        = NULL,
                mar       = c(4, 4.5, 2, 1.5),
                xyswap    = FALSE,
                xlab      = "",
                ylab      = "",
                main      = "",
                xlas      = 0,
                ylas      = 0,
                origin    = FALSE,
                xlim      = NULL,
                ylim      = NULL,
                tol       = .Machine$double.eps ^ 1,
                info      = FALSE,
                maxcount  = 30,
                repair    = TRUE,
                jumpshift = 0.5,
                jumpuphill = min(alpha, 0.01),
                jumpinfo  = FALSE,
                showjump  = FALSE,
                showplot  = TRUE,
                animate   = FALSE,
                delay     = 0.5,
                exact     = FALSE,
                silent    = FALSE )
```

### Arguments

| | |
|---|---|
| `dataset` | a 1 x n vector of data values. |

| | |
|---|---|
| `alpha` | significance level; resulting plot illustrates a 100(1 - `alpha`)% confidence region. |
| `distn` | distribution to fit the dataset to; accepted values: `'cauchy'`, `'gamma'`, `'invgauss'`, `'logis'`, `'llogis'`, `'lnorm'`, `'norm'`, `'unif'`, `'weibull'`. |
| `cen` | a vector of binary values specifying if the corresponding data values are right-censored (0), or observed (1, default); its length must match length(dataset). |
| `heuristic` | numeric value selecting method for plotting: 0 for elliptic-oriented point distribution, and 1 for smoothing boundary search heuristic. |
| `maxdeg` | maximum angle tolerance between consecutive plot segments in degrees. |
| `ellipse_n` | number of roughly equidistant confidence region points to plot using the elliptic-oriented point distribution (must be a multiple of four because its algorithm exploits symmetry in the quadrants of an ellipse). |
| `pts` | displays confidence region boundary points identified if `TRUE`. |
| `mlelab` | logical argument to include the maximum likelihood estimate coordinate point (default is `TRUE`). |
| `sf` | significant figures in axes labels specified using sf = c(x, y), where x and y represent the optional digits argument in the R function `round` as it pertains to the horizontal and vertical labels. |
| `mar` | specifies margin values for `par(mar = c( ))` (see `mar` in `par`). |
| `xyswap` | logical argument to switch the axes that the distribution parameter are shown. |
| `xlab` | string specifying the x axis label. |
| `ylab` | string specifying the y axis label. |
| `main` | string specifying the plot title. |
| `xlas` | numeric in 0, 1, 2, 3 specifying the style of axis labels (see `las` in `par`). |
| `ylas` | numeric in 0, 1, 2, 3 specifying the style of axis labels (see `las` in `par`). |
| `origin` | logical argument to include the plot origin (default is `FALSE`). |
| `xlim` | two-element vector containing horizontal axis minimum and maximum values. |
| `ylim` | two-element vector containing vertical axis minimum and maximum values. |
| `tol` | the `uniroot` parameter specifying its required accuracy. |
| `info` | logical argument to return plot information: MLE is returned as a list; (x, y) plot point coordinates and corresponding phi angles (with respect to MLE) are returned as a list. |
| `maxcount` | integer value specifying the number of smoothing search iterations before terminating with `maxdeg` not met. |
| `repair` | logical argument to repair regions inaccessible using a radial angle from its MLE due to multiple roots at select $\phi$ angles. |
| `jumpshift` | see vignette "conf Advanced Options" for details; location (as a fractional value between 0 and 1) along the vertical or horizontal "gap" (near an uncharted region) to locate a jump-center toward; can be either a scalar value (uniformly applied to all jump-centers) or vector of length four (with unique values for its respective quadrants, relative to the MLE). |

| | |
|---|---|
| `jumpuphill` | see vignette "conf Advanced Options" for details; significance level increase to `alpha` for the jump-center (corresponds to an "uphill" location on its likelihood function); can be either a scalar value (uniformly applied to all jump-centers) or vector of length four (with unique values for its respective quadrants, relative to the MLE). |
| `jumpinfo` | logical argument to return plot info (see `info` argument) and jump-center info; returned within 'repair' attribute are `jumpuphill` value, `jumpshift` value, "\|" or "-" gap type, jump-center(s) coordinates, and coordinates of points left & right of the inaccessible region. |
| `showjump` | logical argument specifying if jump-center repair reference points appear on the confidence region plot. |
| `showplot` | logical argument specifying if a plot is output; altering from its default of `TRUE` is only logical assuming `crplot` is run for its data only (see the `info` argument). |
| `animate` | logical argument specifying if an animated plot build will display; the annimation sequence is given in successive plots. |
| `delay` | numeric value of delay (in seconds) between successive plots when `animate = TRUE`. |
| `exact` | logical argument specifying if alpha value is adjusted to compensate for negative coverage bias to achieve (1 - alpha) coverage probability using previously recorded Monte Carlo simulation results; available for limited values of alpha (roughly <= 0.2–0.3), n (typically n = 4, 5, ..., 50) and distributions (distn suffixes: weibull, llogis, norm). |
| `silent` | logical argument specifying if console output should be suppressed. |

## Details

This function plots a confidence region for a variety of two-parameter distributions. It requires:

- a vector of dataset values,
- the level of significance (alpha), and
- a population distribution to fit the data to.

Plots display according to probability density function parameterization given later in this section. Two heuristics (and their associated combination) are available to plot confidence regions. Along with their descriptions, they are:

1. *Smoothing Boundary Search Heuristic (default).* This heuristic plots more points in areas of greater curvature to ensure a smooth appearance throughout the confidence region boundary. Its `maxdeg` parameter specifies the maximum tolerable angle between three successive points. Lower values of `maxdeg` result in smoother plots, and its default value of 5 degrees provides adequate smoothing in most circumstances. Values of `maxdeg` $\leq 3$ are not recommended due to their complicating implications to trigonometric numerical approximations near 0 and 1; their use may result in plot errors.
2. *Elliptic-Oriented Point Distribution.* This heuristic allows the user to specify a number of points to plot along the confidence region boundary at roughly uniform intervals. Its name is derived from the technique it uses to choose these points—an extension of the Steiner generation of a non-degenerate conic section, also known as the parallelogram method—which

identifies points along an ellipse that are approximately equidistant. To exploit the computational benefits of ellipse symmetry over its four quadrants, `ellipse_n` value must be divisible by four.

By default, `crplot` implements the smoothing boundary search heuristic. Alternatively, the user can plot using the elliptic-oriented point distribution algorithm, or a combination of them both. Combining the two techniques initializes the plot using the elliptic-oriented point distribution algorithm, and then subsequently populates additional points in areas of high curvature (those outside of the maximum angle tolerance parameterization) in accordance with the smoothing boundary search heuristic. This combination results when the smoothing boundary search heuristic is specified in conjunction with an `ellipse_n` value greater than four.

Both of the aforementioned heuristics use a radial profile log likelihood function to identify points along the confidence region boundary. It cuts the log likelihood function in a directional azimuth from its MLE, and locates the associated confidence region boundary point using the asymptotic results associated with the ratio test statistic $-2[logL(\theta) - logL(\theta hat)]$ which converges in distribution to the chi-square distribution with two degrees of freedom (for a two parameter distribution).

The default axes convention in use by `crplot` are

| | Horizontal | Vertical |
|---|---|---|
| Distribution | Axis | Axis |
| Cauchy | $a$ | $s$ |
| gamma | $\theta$ | $\kappa$ |
| inverse Gaussian | $\mu$ | $\lambda$ |
| log logistic | $\lambda$ | $\kappa$ |
| log normal | $\mu$ | $\sigma$ |
| logistic | $\mu$ | $\sigma$ |
| normal | $\mu$ | $\sigma$ |
| uniform | $a$ | $b$ |
| Weibull | $\kappa$ | $\lambda$ |

where each respective distribution is defined below.

- The Cauchy distribution for the real-numbered location parameter $a$, scale parameter $s$, and $x$ is a real number, has the probability density function
$$1/(s\pi(1 + ((x - a)/s)^2)).$$
- The gamma distribution for shape parameter $\kappa > 0$, scale parameter $\theta > 0$, and $x > 0$, has the probability density function
$$1/(Gamma(\kappa)\theta^\kappa)x^{(\kappa-1)}exp(-x/\theta).$$
- The inverse Gaussian distribution for mean $\mu > 0$, shape parameter $\lambda > 0$, and $x > 0$, has the probability density function
$$\sqrt(\lambda/(2\pi x^3))exp(-\lambda(x - \mu)^2/(2\mu^2 x)).$$
- The log logistic distribution for scale parameter $\lambda > 0$, shape parameter $\kappa > 0$, and $x \geq 0$, has a probability density function
$$(\kappa\lambda)(x\lambda)^{(\kappa-1)}/(1 + (\lambda x)^\kappa)^2.$$

- The log normal distribution for the real-numbered mean $\mu$ of the logarithm, standard deviation $\sigma > 0$ of the logarithm, and $x > 0$, has the probability density function
$$1/(x\sigma\sqrt(2\pi))exp(-(\log x - \mu)^2/(2\sigma^2)).$$
- The logistic distribution for the real-numbered location parameter $\mu$, scale parameter $\sigma$, and $x$ is a real number, has the probability density function
$$(1/\sigma)exp((x-\mu)/\sigma)(1+exp((x-\mu)/\sigma))^{-2}$$
- The normal distribution for the real-numbered mean $\mu$, standard deviation $\sigma > 0$, and $x$ is a real number, has the probability density function
$$1/\sqrt(2\pi\sigma^2)exp(-(x-\mu)^2/(2\sigma^2)).$$
- The uniform distribution for real-valued parameters $a$ and $b$ where $a < b$ and $a \le x \le b$, has the probability density function
$$1/(b-a).$$
- The Weibull distribution for scale parameter $\lambda > 0$, shape parameter $\kappa > 0$, and $x > 0$, has the probability density function
$$\kappa(\lambda^\kappa)x^{(\kappa-1)}exp(-(\lambda x)^\kappa).$$

## Value

If the optional argument `info = TRUE` is included then a list is returned with:

- parm1*: a vector containing the associated confidence region boundary values for parameter 1
- parm2*: a vector containing the associated confidence region boundary values for parameter 2
- phi: a vector containing the angles used
- parm1hat*: the MLE for parameter 1
- parm2hat*: the MLE for parameter 2

*Note: "param1" and "param2" are placeholders that will be replaced with the appropriate parameter names based on the probability distribution.

## Author(s)

Christopher Weld (<ceweld@email.wm.edu>)

Lawrence Leemis (<leemis@math.wm.edu>)

## References

Jaeger, A. (2016), "Computation of Two- and Three-Dimensional Confidence Regions with the Likelihood Ratio", The American Statistician, 49, 48–53.

Weld, C., Loh, A., Leemis, L. (in press), "Plotting Likelihood-Ratio Based Confidence Regions for Two-Parameter Univariate Probability Models", The American Statistician.

### See Also

coversim, uniroot

### Examples

```
## plot the 95% confidence region for Weibull shape and scale parameters
## corresponding to the given ballbearing dataset
ballbearing <- c(17.88, 28.92, 33.00, 41.52, 42.12, 45.60, 48.48, 51.84,
                 51.96, 54.12, 55.56, 67.80, 68.64, 68.64, 68.88, 84.12,
                 93.12, 98.64, 105.12, 105.84, 127.92, 128.04, 173.40)
crplot(dataset = ballbearing, distn = "weibull", alpha = 0.05)

## repeat this plot using the elliptic-oriented point distribution heuristic
crplot(dataset = ballbearing, distn = "weibull", alpha = 0.05,
       heuristic = 0, ellipse_n = 80)

## combine the two heuristics, compensating any elliptic-oriented point verticies whose apparent
## angles > 6 degrees with additional points, and expand the plot area to include the origin
crplot(dataset = ballbearing, distn = "weibull", alpha = 0.05,
       maxdeg = 6, ellipse_n = 80, origin = TRUE)

## next use the inverse Gaussian distribution and show no plot points
crplot(dataset = ballbearing, distn = "invgauss", alpha = 0.05,
       pts = FALSE)
```

## dinvgauss *The Inverse Gaussian Distribution*

### Description

Density, distribution function, quantile function, and random generation for the inverse Gaussian. The corresponding code for these functions as well as the manual information included here is attributed to Christophe Pouzat's STAR Package (archived 2022-05-23).

### Usage

```
dinvgauss(x, mu = 1, sigma2 = 1, boundary = NULL,
          log = FALSE)
pinvgauss(q, mu = 1, sigma2 = 1,
          boundary = NULL, lower.tail = TRUE,
          log.p = FALSE)
qinvgauss(p, mu = 1, sigma2 = 1,
          boundary = NULL)
rinvgauss(n = 1, mu = 1, sigma2 = 1, boundary = NULL)
```

## Arguments

| | |
|---|---|
| `x, q` | vector of quantiles. |
| `p` | vector of probabilities. |
| `n` | number of observations. If `length(n) > 1`, the length is taken to be the number required. |
| `mu` | mean value of the distribution in the default parameterization, `mean value / boundary` otherwise. Can also be viewed as the inverse of the drift of the latent Brownian motion. |
| `sigma2` | variance of the latent Brownian motion. When this parameterization is used (the default) the distance between the "starting" point and the boundary ("absorbing barrier") is set to 1. |
| `boundary` | distance between the starting point and the "absorbing barrier" of the latent Brownian motion. When this parameterization is used the Brownian motion variance is set to 1. |
| `lower.tail` | logical; if `TRUE` (default), probabilities are `P[X <= x]`, otherwise, `P[X > x]`. |
| `log, log.p` | logical; if `TRUE`, probabilities p are given as log(p). |

## Details

With the default, `"sigma2"`, parameterization (`mu = m, sigma2 = s^2`) the inverse Gaussian distribution has density:

$$f(x) = \frac{1}{\sqrt{2\,\pi\,\sigma^2\,x^3}}\ \exp(-\frac{1}{2}\frac{(x-\mu)^2}{x\,\sigma^2\,\mu^2})$$

with $\sigma^2 > 0$. The theoretical mean is: $\mu$ and the theoretical variance is: $\mu^3\sigma^2$. With the default, `"boundary"`, parameterization (`mu = m, boundary = b`)the inverse Gaussian distribution has density:

$$f(x) = \frac{b}{\sqrt{2\,\pi\,x^3}}\ \exp(-\frac{1}{2}\frac{(x-b\,\mu)^2}{x\,\mu^2})$$

with $\sigma^2 > 0$. The theoretical mean is: $\mu\,b$ and the theoretical variance is: $\mu^3\sigma^2$. The latent Brownian motion is described in Lindsey (2004) pp 209-213, Whitemore and Seshadri (1987), Aalen and Gjessing (2001) and Gerstein and Mandelbrot (1964).

The expression for the distribution function is given in Eq. 4 of Whitemore and Seshadri (1987).

Initial guesses for the inversion of the distribution function used in `qinvgauss` are obtained with the transformation of Whitemore and Yalovsky (1978).

Random variates are obtained with the method of Michael et al (1976) which is also described by Devroye (1986, p 148) and Gentle (2003, p 193).

## Value

`dinvgauss` gives the density, `pinvgauss` gives the distribution function, `qinvgauss` gives the quantile function and `rinvgauss` generates random deviates.

## Author(s)

Christophe Pouzat <`christophe.pouzat@gmail.com`>

## References

Gerstein, George L. and Mandelbrot, Benoit (1964) Random Walk Models for the Spike Activity of a Single Neuron. *Biophys J.* **4**: 41–68.

Whitemore, G. A. and Yalovsky, M. (1978) A normalizing logarithmic transformation for inverse Gaussian random variables. *Technometrics* **20**: 207–208.

Whitmore, G. A. and Seshadri, V. (1987) A Heuristic Derivation of the Inverse Gaussian Distribution. *The American Statistician* **41**: 280–281.

Aalen, Odd O. and Gjessing, Hakon K. (2001) Understanding the Shape of the Hazard Rate: A Process Point of View. *Statistical Science* **16**: 1–14.

Lindsey, J.K. (2004) *Introduction to Applied Statistics: A Modelling Approach*. OUP.

Michael, J. R., Schucany, W. R. and Haas, R. W. (1976) Generating random variates using transformations with multiple roots. *The American Statistician* **30**: 88–90.

Devroye, L. (1986) *Non-Uniform Random Variate Generation*. Springer-Verlag.

Gentle, J. E. (2003) *Random Number Generation and Monte Carlo Methods*. Springer.

## See Also

`invgaussMLE`.

## Examples

```
## Not run:
## Start with the inverse Gauss
## Define standard mu and sigma
mu.true <- 0.075 ## a mean ISI of 75 ms
sigma2.true <- 3
## Define a sequence of points on the time axis
X <- seq(0.001,0.3,0.001)
## look at the density
plot(X,dinvgauss(X,mu.true,sigma2.true),type="l",xlab="ISI (s)",ylab="Density")

## Generate a sample of 100 ISI from this distribution
sampleSize <- 100
sampIG <- rinvgauss(sampleSize,mu=mu.true,sigma2=sigma2.true)
## check out the empirical survival function (obtained with the Kaplan-Meyer
## estimator) against the true one
library(survival)
sampIG.KMfit <- survfit(Surv(sampIG,1+numeric(length(sampIG))) ~1)
plot(sampIG.KMfit,log=TRUE)
lines(X,pinvgauss(X,mu.true,sigma2.true,lower.tail=FALSE),col=2)

## Get a ML fit
sampIGmleIG <- invgaussMLE(sampIG)
## compare true and estimated parameters
rbind(est = sampIGmleIG$estimate,se = sampIGmleIG$se,true = c(mu.true,sigma2.true))
## plot contours of the log relative likelihood function
Mu <- seq(sampIGmleIG$estimate[1]-3*sampIGmleIG$se[1],
          sampIGmleIG$estimate[1]+3*sampIGmleIG$se[1],
          sampIGmleIG$se[1]/10)
Sigma2 <- seq(sampIGmleIG$estimate[2]-7*sampIGmleIG$se[2],
              sampIGmleIG$estimate[2]+7*sampIGmleIG$se[2],
              sampIGmleIG$se[2]/10)
sampIGmleIGcontour <- sapply(Mu, function(mu) sapply(Sigma2, function(s2) sampIGmleIG$r(mu,s2)))
contour(Mu,Sigma2,t(sampIGmleIGcontour),
        levels=c(log(c(0.5,0.1)),-0.5*qchisq(c(0.95,0.99),df=2)),
        labels=c("log(0.5)",
          "log(0.1)",
          "-1/2*P(Chi2=0.95)",
          "-1/2*P(Chi2=0.99)"),
        xlab=expression(mu),ylab=expression(sigma^2))
points(mu.true,sigma2.true,pch=16,col=2)
## We can see that the contours are more parabola like on a log scale
contour(log(Mu),log(Sigma2),t(sampIGmleIGcontour),
        levels=c(log(c(0.5,0.1)),-0.5*qchisq(c(0.95,0.99),df=2)),
        labels=c("log(0.5)",
          "log(0.1)",
          "-1/2*P(Chi2=0.95)",
          "-1/2*P(Chi2=0.99)"),
        xlab=expression(log(mu)),ylab=expression(log(sigma^2)))
points(log(mu.true),log(sigma2.true),pch=16,col=2)
## make a deviance test for the true parameters
pchisq(-2*sampIGmleIG$r(mu.true,sigma2.true),df=2)
## check fit with a QQ plot
qqDuration(sampIGmleIG,log="xy")

## Generate a censored sample using an exponential distribution
sampEXP <- rexp(sampleSize,1/(2*mu.true))
sampIGtime <- pmin(sampIG,sampEXP)
sampIGstatus <- as.numeric(sampIG <= sampEXP)
## fit the censored sample
sampIG2mleIG <- invgaussMLE(sampIGtime,,sampIGstatus)
## look at the results
rbind(est = sampIG2mleIG$estimate,
      se = sampIG2mleIG$se,
      true = c(mu.true,sigma2.true))
pchisq(-2*sampIG2mleIG$r(mu.true,sigma2.true),df=2)
## repeat the survival function estimation
sampIG2.KMfit <- survfit(Surv(sampIGtime,sampIGstatus) ~1)
plot(sampIG2.KMfit,log=TRUE)
lines(X,pinvgauss(X,sampIG2mleIG$estimate[1],sampIG2mleIG$estimate[2],lower.tail=FALSE),col=2)

## End(Not run)
```

## dllogis *The Log Logistic Distribution*

## Description

Density, distribution function, quantile function, and random generation for the log logistic. The corresponding code for these functions as well as the manual information included here is attributed to Christophe Pouzat's STAR Package (archived 2022-05-23).

## Usage

```
dllogis(x, location = 0, scale = 1, log = FALSE)
pllogis(q, location = 0, scale = 1, lower.tail = TRUE, log.p = FALSE)
qllogis(p, location = 0, scale = 1, lower.tail = TRUE, log.p = FALSE)
rllogis(n, location = 0, scale = 1)
```

## Arguments

| | |
|---|---|
| `x, q` | vector of quantiles. |
| `p` | vector of probabilities. |
| `n` | number of observations. If `length(n) > 1`, the length is taken to be the number required. |
| `location, scale` | location and scale parameters (non-negative numeric). |
| `lower.tail` | logical; if `TRUE` (default), probabilities are `P[X <= x]`, otherwise, `P[X > x]`. |
| `log, log.p` | logical; if `TRUE`, probabilities p are given as log(p). |

## Details

If `location` or `scale` are omitted, they assume the default values of 0 and 1 respectively.

The log-Logistic distribution with `location = m` and `scale = s` has distribution function

$$\mathrm{F}(x) = \frac{1}{1 + \exp(-\frac{\log(x) - m}{s})}$$

and density

$$\mathrm{f}(x) = \frac{1}{s\,x} \frac{\exp(-\frac{\log(x) - m}{s})}{(1 + \exp(-\frac{\log(x) - m}{s}))^2}$$

## Value

`dllogis` gives the density, `pllogis` gives the distribution function, `qllogis` gives the quantile function and `rllogis` generates random deviates.

## Author(s)

Christophe Pouzat <`christophe.pouzat@gmail.com`>

## References

Lindsey, J.K. (2004) *Introduction to Applied Statistics: A Modelling Approach*. OUP.

Lindsey, J.K. (2004) *The Statistical Analysis of Stochastic Processes in Time*. CUP.

## See Also

llogisMLE.

## Examples

```
## Not run:
tSeq <- seq(0.001,0.6,0.001)
location.true <- -2.7
scale.true <- 0.025
Yd <- dllogis(tSeq, location.true, scale.true)
Yh <- hllogis(tSeq, location.true, scale.true)
max.Yd <- max(Yd)
max.Yh <- max(Yh)
Yd <- Yd / max.Yd
Yh <- Yh / max.Yh
oldpar <- par(mar=c(5,4,4,4))
plot(tSeq, Yd, type="n", axes=FALSE, ann=FALSE,
     xlim=c(0,0.6), ylim=c(0,1))
axis(2,at=seq(0,1,0.2),labels=round(seq(0,1,0.2)*max.Yd,digits=2))
mtext("Density (1/s)", side=2, line=3)
axis(1,at=pretty(c(0,0.6)))
mtext("Time (s)", side=1, line=3)
axis(4, at=seq(0,1,0.2), labels=round(seq(0,1,0.2)*max.Yh,digits=2))
mtext("Hazard (1/s)", side=4, line=3, col=2)
mtext("Log Logistic Density and Hazard Functions", side=3, line=2,cex=1.5)
lines(tSeq,Yd)
lines(tSeq,Yh,col=2)
par(oldpar)

## End(Not run)
```

---

`gammaMLE` *Maximum Likelihood Parameter Estimation of a Gamma Model with Possibly Censored Data*

---

## Description

Estimate Gamma model parameters by the maximum likelihood method using possibly censored data. Two different parameterizations of the Gamma distribution can be used. The corresponding code for this function as well as the manual information included here is attributed to Christophe Pouzat's STAR Package (archived 2022-05-23).

## Usage

```
gammaMLE(yi, ni = numeric(length(yi)) + 1,
         si = numeric(length(yi)) + 1, scale = TRUE)
```

## Arguments

| | |
|---|---|
| `yi` | vector of (possibly binned) observations or a `spikeTrain` object. |
| `ni` | vector of counts for each value of `yi`; default: `numeric(length(yi))+1`. |
| `si` | vector of counts of *uncensored* observations for each value of `yi`; default: `numeric(length(yi))+1`. |
| `scale` | logical should the scale (`TRUE`) or the rate parameterization (`FALSE`) be used? |

## Details

There is no closed form expression for the MLE of a Gamma distribution. The numerical method implemented here uses the profile likelihood described by Monahan (2001) pp 210-216.

In order to ensure good behavior of the numerical optimization routines, optimization is performed on the log of the parameters (`shape` and `scale` or `rate`).

Standard errors are obtained from the inverse of the observed information matrix at the MLE. They are transformed to go from the log scale used by the optimization routine to the parameterization requested.

## Value

A list of class `durationFit` with the following components:

| | |
|---|---|
| `estimate` | the estimated parameters, a named vector. |
| `se` | the standard errors, a named vector. |
| `logLik` | the log likelihood at maximum. |
| `r` | a function returning the log of the relative likelihood function. |
| `mll` | a function returning the opposite of the log likelihood function using the log of the parameters. |
| `call` | the matched call. |

## Note

The returned standard errors (component `se`) are valid in the asymptotic limit. You should plot contours using function `r` in the returned list and check that the contours are reasonably close to ellipses.

## Author(s)

Christophe Pouzat <`christophe.pouzat@gmail.com`>

## References

Monahan, J. F. (2001) *Numerical Methods of Statistics*. CUP.

Lindsey, J.K. (2004) *Introduction to Applied Statistics: A Modelling Approach*. OUP.

## See Also

invgaussMLE.

## Examples

```
## Not run:
## Simulate sample of size 100 from a gamma distribution
set.seed(1102006,"Mersenne-Twister")
sampleSize <- 100
shape.true <- 6
scale.true <- 0.012
sampGA <- rgamma(sampleSize,shape=shape.true,scale=scale.true)
sampGAmleGA <- gammaMLE(sampGA)
rbind(est = sampGAmleGA$estimate,se = sampGAmleGA$se,true = c(shape.true,scale.true))

## Estimate the log relative likelihood on a grid to plot contours
Shape <- seq(sampGAmleGA$estimate[1]-4*sampGAmleGA$se[1],
               sampGAmleGA$estimate[1]+4*sampGAmleGA$se[1],
               sampGAmleGA$se[1]/10)
Scale <- seq(sampGAmleGA$estimate[2]-4*sampGAmleGA$se[2],
             sampGAmleGA$estimate[2]+4*sampGAmleGA$se[2],
             sampGAmleGA$se[2]/10)
sampGAmleGAcontour <- sapply(Shape, function(sh) sapply(Scale, function(sc) sampGAmleGA$r(sh,sc)))
## plot contours using a linear scale for the parameters
## draw four contours corresponding to the following likelihood ratios:
##  0.5, 0.1, Chi2 with 2 df and p values of 0.95 and 0.99
X11(width=12,height=6)
layout(matrix(1:2,ncol=2))
contour(Shape,Scale,t(sampGAmleGAcontour),
        levels=c(log(c(0.5,0.1)),-0.5*qchisq(c(0.95,0.99),df=2)),
        labels=c("log(0.5)",
          "log(0.1)",
          "-1/2*P(Chi2=0.95)",
          "-1/2*P(Chi2=0.99)"),
        xlab="shape",ylab="scale",
        main="Log Relative Likelihood Contours"
        )
points(sampGAmleGA$estimate[1],sampGAmleGA$estimate[2],pch=3)
points(shape.true,scale.true,pch=16,col=2)
## The contours are not really symmetrical about the MLE we can try to
## replot them using a log scale for the parameters to see if that improves
## the situation
contour(log(Shape),log(Scale),t(sampGAmleGAcontour),
        levels=c(log(c(0.5,0.1)),-0.5*qchisq(c(0.95,0.99),df=2)),
        labels="",
        xlab="log(shape)",ylab="log(scale)",
        main="Log Relative Likelihood Contours",
        sub="log scale for the parameters")
points(log(sampGAmleGA$estimate[1]),log(sampGAmleGA$estimate[2]),pch=3)
points(log(shape.true),log(scale.true),pch=16,col=2)

## make a parametric boostrap to check the distribution of the deviance
```

```
nbReplicate <- 10000
sampleSize <- 100
system.time(
            devianceGA100 <- replicate(nbReplicate,{
                           sampGA <- rgamma(sampleSize,shape=shape.true,scale=scale.true)
                              sampGAmleGA <- gammaMLE(sampGA)
                              -2*sampGAmleGA$r(shape.true,scale.true)
                            }
                                        )
            )[3]

## Get 95 and 99% confidence intervals for the QQ plot
ci <- sapply(1:nbReplicate,
                 function(idx) qchisq(qbeta(c(0.005,0.025,0.975,0.995),
                                            idx,
                                            nbReplicate-idx+1),
                                      df=2)
             )
## make QQ plot
X <- qchisq(ppoints(nbReplicate),df=2)
Y <- sort(devianceGA100)
X11()
plot(X,Y,type="n",
     xlab=expression(paste(chi[2]^2," quantiles")),
     ylab="MC quantiles",
     main="Deviance with true parameters after ML fit of gamma data",
     sub=paste("sample size:", sampleSize,"MC replicates:", nbReplicate)
     )
abline(a=0,b=1)
lines(X,ci[1,],lty=2)
lines(X,ci[2,],lty=2)
lines(X,ci[3,],lty=2)
lines(X,ci[4,],lty=2)
lines(X,Y,col=2)

## End(Not run)
```

---

## `invgaussMLE` *Maximum Likelihood Parameter Estimation of an Inverse Gaussian Model with Possibly Censored Data*

---

### Description

Estimate inverse Gaussian model parameters by the maximum likelihood method using possibly censored data. Two different parameterizations of the inverse Gaussian distribution can be used. The corresponding code for this function as well as the manual information included here is attributed to Christophe Pouzat's STAR Package (archived 2022-05-23).

## Usage

```
invgaussMLE(yi, ni = numeric(length(yi)) + 1,
            si = numeric(length(yi)) + 1,
            parameterization = "sigma2")
```

## Arguments

| | |
|---|---|
| `yi` | vector of (possibly binned) observations or a `spikeTrain` object. |
| `ni` | vector of counts for each value of `yi`; default: `numeric(length(yi))+1`. |
| `si` | vector of counts of *uncensored* observations for each value of `yi`; default: `numeric(length(yi))+1`. |
| `parameterization` | parameterization used, `"sigma2"` (default) of `"boundary"`. |

## Details

The 2 different parameterizations of the inverse Gaussian distribution are discussed in the manual of `dinvgauss`.

In the absence of censored data the ML estimates are available in closed form (Lindsey, 2004, p 212) together with the Hessian matrix at the MLE. In presence of censored data an initial guess for the parameters is obtained using the uncensored data before maximizing the likelihood function to the full data set using `optim` with the `BFGS` method. ML estimation is always performed with the `"sigma2"` parameterization. Parameters and variance-covariance matrix are transformed at the end if the `"boundary"` parameterization is requested.

In order to ensure good behavior of the numerical optimization routines, optimization is performed on the log of the parameters (`mu` and `sigma2`).

Standard errors are obtained from the inverse of the observed information matrix at the MLE. They are transformed to go from the log scale used by the optimization routine, when the latter is used (ie, for censored data) to the parameterization requested.

## Value

A list of class `durationFit` with the following components:

| | |
|---|---|
| `estimate` | the estimated parameters, a named vector. |
| `se` | the standard errors, a named vector. |
| `logLik` | the log likelihood at maximum. |
| `r` | a function returning the log of the relative likelihood function. |
| `mll` | a function returning the opposite of the log likelihood function using the log of the parameters. |
| `call` | the matched call. |

## Note

The returned standard errors (component `se`) are valid in the asymptotic limit. You should plot contours using function `r` in the returned list and check that the contours are reasonably close to ellipses.

## Author(s)

Christophe Pouzat <christophe.pouzat@gmail.com>

## References

Lindsey, J.K. (2004) *Introduction to Applied Statistics: A Modelling Approach*. OUP.

## See Also

dinvgauss,gammaMLE,llogisMLE.

## Examples

```
## Simulate sample of size 100 from an inverse Gaussian
## distribution
set.seed(1102006,"Mersenne-Twister")
sampleSize <- 100
mu.true <- 0.075
sigma2.true <- 3
sampleSize <- 100
sampIG <- rinvgauss(sampleSize,mu=mu.true,sigma2=sigma2.true)
## Make a maximum likelihood fit
sampIGmleIG <- invgaussMLE(sampIG)
## Compare estimates with actual values
rbind(est = coef(sampIGmleIG),se = sampIGmleIG$se,true = c(mu.true,sigma2.true))
## In the absence of censoring the MLE of the inverse Gaussian is available in a
## closed form together with its variance (ie, the observed information matrix)
## we can check that we did not screw up at that stage by comparing the observed
## information matrix obtained numerically with the analytical one. To do that we
## use the MINUS log likelihood function returned by invgaussMLE to get a numerical
## estimate
detailedFit <- optim(par=as.vector(log(sampIGmleIG$estimate)),
                     fn=sampIGmleIG$mll,
                     method="BFGS",
                     hessian=TRUE)
## We should not forget that the "mll" function uses the log of the parameters while
## the "se" component of sampIGmleIG list is expressed on the linear scale we must therefore
## transform one into the other as follows (Kalbfleisch, 1985, p71):
## if x = exp(u) and y = exp(v) and if we have the information matrix in term of
## u and v (that's the hessian component of list detailedFit above), we create matrix:
##      du/dx du/dy
## Q =
##      dv/dx dv/dy
## and we get I in term of x and y by the following matrix product:
## I(x,y) <- t(Q) %*% I(u,v) %*% Q
## In the present case:
##  du/dx = 1/exp(u), du/dy = 0, dv/dx = 0, dv/dy = 1/exp(v)
## Therefore:
Q <- diag(1/exp(detailedFit$par))
numericalI <- t(Q) %*% detailedFit$hessian %*% Q
seComp <- rbind(sampIGmleIG$se, sqrt(diag(solve(numericalI))))
colnames(seComp) <- c("mu","sigma2")
```

```
rownames(seComp) <- c("analytical", "numerical")
seComp
## We can check the relative differences between the 2
apply(seComp,2,function(x) abs(diff(x))/x[1])

## Not run:
## Estimate the log relative likelihood on a grid to plot contours
Mu <- seq(coef(sampIGmleIG)[1]-4*sampIGmleIG$se[1],
          coef(sampIGmleIG)[1]+4*sampIGmleIG$se[1],
          sampIGmleIG$se[1]/10)
Sigma2 <- seq(coef(sampIGmleIG)[2]-4*sampIGmleIG$se[2],
              coef(sampIGmleIG)[2]+4*sampIGmleIG$se[2],
              sampIGmleIG$se[2]/10)
sampIGmleIGcontour <- sapply(Mu, function(mu) sapply(Sigma2, function(s2) sampIGmleIG$r(mu,s2)))
## plot contours using a linear scale for the parameters
## draw four contours corresponding to the following likelihood ratios:
##  0.5, 0.1, Chi2 with 2 df and p values of 0.95 and 0.99
X11(width=12,height=6)
layout(matrix(1:2,ncol=2))
contour(Mu,Sigma2,t(sampIGmleIGcontour),
        levels=c(log(c(0.5,0.1)),-0.5*qchisq(c(0.95,0.99),df=2)),
        labels=c("log(0.5)",
          "log(0.1)",
          "-1/2*P(Chi2=0.95)",
          "-1/2*P(Chi2=0.99)"),
        xlab=expression(mu),ylab=expression(sigma^2),
        main="Log Relative Likelihood Contours"
        )
points(coef(sampIGmleIG)[1],coef(sampIGmleIG)[2],pch=3)
points(mu.true,sigma2.true,pch=16,col=2)
## The contours are not really symmetrical about the MLE we can try to
## replot them using a log scale for the parameters to see if that improves
## the situation
contour(log(Mu),log(Sigma2),t(sampIGmleIGcontour),
        levels=c(log(c(0.5,0.1)),-0.5*qchisq(c(0.95,0.99),df=2)),
        labels="",
        xlab=expression(log(mu)),ylab=expression(log(sigma^2)),
        main="Log Relative Likelihood Contours",
        sub="log scale for the parameters")
points(log(coef(sampIGmleIG)[1]),log(coef(sampIGmleIG)[2]),pch=3)
points(log(mu.true),log(sigma2.true),pch=16,col=2)

## Even with the log scale the contours are not ellipsoidal, so let us compute profiles
## For that we are going to use the returned MINUS log likelihood function
logMuProfFct <- function(logMu,...) {
  myOpt <- optimise(function(x) sampIGmleIG$mll(c(logMu,x))+logLik(sampIGmleIG),...)
  as.vector(unlist(myOpt[c("objective","minimum")]))
}
logMuProfCI <- function(logMu,
                        CI,
                        a=logS2Seq[1],
                b=logS2Seq[length(logS2Seq)]) logMuProfFct(logMu,c(a,b))[1] - qchisq(CI,1)/2
```

```
logS2ProfFct <- function(logS2,...) {
  myOpt <- optimise(function(x) sampIGmleIG$mll(c(x,logS2))+logLik(sampIGmleIG),...)
  as.vector(unlist(myOpt[c("objective","minimum")]))
}
logS2ProfCI <- function(logS2, CI,
                        a=logMuSeq[1],
                b=logMuSeq[length(logMuSeq)]) logS2ProfFct(logS2,c(a,b))[1] - qchisq(CI,1)/2


## We compute profiles (on the log scale) eploxing +/- 3 times
## the se about the MLE
logMuSE <- sqrt(diag(solve(detailedFit$hessian)))[1]
logMuSeq <- seq(log(coef(sampIGmleIG)[1])-3*logMuSE,
                log(coef(sampIGmleIG)[1])+3*logMuSE,
                logMuSE/10)
logS2SE <- sqrt(diag(solve(detailedFit$hessian)))[2]
logS2Seq <- seq(log(coef(sampIGmleIG)[2])-3*logS2SE,
                log(coef(sampIGmleIG)[2])+3*logS2SE,
                logS2SE/10)
logMuProf <- sapply(logMuSeq,logMuProfFct,
                    lower=logS2Seq[1],
                    upper=logS2Seq[length(logS2Seq)])
## Get 95
logMuCI95 <- c(uniroot(logMuProfCI,
                       interval=c(logMuSeq[1],log(coef(sampIGmleIG)[1])),
                       CI=0.95)$root,
               uniroot(logMuProfCI,
                       interval=c(log(coef(sampIGmleIG)[1]),logMuSeq[length(logMuSeq)]),
                       CI=0.95)$root
               )
logMuCI99 <- c(uniroot(logMuProfCI,
                       interval=c(logMuSeq[1],log(coef(sampIGmleIG)[1])),
                       CI=0.99)$root,
               uniroot(logMuProfCI,
                       interval=c(log(coef(sampIGmleIG)[1]),logMuSeq[length(logMuSeq)]),
                       CI=0.99)$root
               )

logS2Prof <- sapply(logS2Seq,logS2ProfFct,
                    lower=logMuSeq[1],
                    upper=logMuSeq[length(logMuSeq)])
## Get 95
logS2CI95 <- c(uniroot(logS2ProfCI,
                       interval=c(logS2Seq[1],log(coef(sampIGmleIG)[2])),
                       CI=0.95)$root,
               uniroot(logS2ProfCI,
                       interval=c(log(coef(sampIGmleIG)[2]),logS2Seq[length(logS2Seq)]),
                       CI=0.95)$root
               )
logS2CI99 <- c(uniroot(logS2ProfCI,
                       interval=c(logS2Seq[1],log(coef(sampIGmleIG)[2])),
                       CI=0.99)$root,
               uniroot(logS2ProfCI,
                       interval=c(log(coef(sampIGmleIG)[2]),logS2Seq[length(logS2Seq)]),
                       CI=0.99)$root
               )


## Add profiles to the previous plot
lines(logMuSeq,logMuProf[2,],col=2,lty=2)
lines(logS2Prof[2,],logS2Seq,col=2,lty=2)

## We can now check the deviations of the (profiled) deviances
## from the asymptotic parabolic curves
X11()
layout(matrix(1:4,nrow=2))
oldpar <- par(mar=c(4,4,2,1))
logMuSeqOffset <- logMuSeq-log(coef(sampIGmleIG)[1])
logMuVar <- diag(solve(detailedFit$hessian))[1]
plot(logMuSeq,2*logMuProf[1,],type="l",xlab=expression(log(mu)),ylab="Deviance")
lines(logMuSeq,logMuSeqOffset^2/logMuVar,col=2)
points(log(coef(sampIGmleIG)[1]),0,pch=3)
abline(h=0)
abline(h=qchisq(0.95,1),lty=2)
abline(h=qchisq(0.99,1),lty=2)
lines(rep(logMuCI95[1],2),c(0,qchisq(0.95,1)),lty=2)
lines(rep(logMuCI95[2],2),c(0,qchisq(0.95,1)),lty=2)
lines(rep(logMuCI99[1],2),c(0,qchisq(0.99,1)),lty=2)
lines(rep(logMuCI99[2],2),c(0,qchisq(0.99,1)),lty=2)
## We can also "linearize" this last graph
plot(logMuSeq,
     sqrt(2*logMuProf[1,])*sign(logMuSeqOffset),
     type="l",
     xlab=expression(log(mu)),
     ylab=expression(paste("signed ",sqrt(Deviance)))
     )
lines(logMuSeq,
      sqrt(logMuSeqOffset^2/logMuVar)*sign(logMuSeqOffset),
      col=2)
points(log(coef(sampIGmleIG)[1]),0,pch=3)

logS2SeqOffset <- logS2Seq-log(coef(sampIGmleIG)[2])
logS2Var <- diag(solve(detailedFit$hessian))[2]
plot(logS2Seq,2*logS2Prof[1,],type="l",xlab=expression(log(sigma^2)),ylab="Deviance")
lines(logS2Seq,logS2SeqOffset^2/logS2Var,col=2)
points(log(coef(sampIGmleIG)[2]),0,pch=3)
abline(h=0)
abline(h=qchisq(0.95,1),lty=2)
abline(h=qchisq(0.99,1),lty=2)
lines(rep(logS2CI95[1],2),c(0,qchisq(0.95,1)),lty=2)
lines(rep(logS2CI95[2],2),c(0,qchisq(0.95,1)),lty=2)
lines(rep(logS2CI99[1],2),c(0,qchisq(0.99,1)),lty=2)
lines(rep(logS2CI99[2],2),c(0,qchisq(0.99,1)),lty=2)
## We can also "linearize" this last graph
plot(logS2Seq,
     sqrt(2*logS2Prof[1,])*sign(logS2SeqOffset),
     type="l",
     xlab=expression(log(sigma^2)),
     ylab=expression(paste("signed ",sqrt(Deviance)))
     )
lines(logS2Seq,
      sqrt(logS2SeqOffset^2/logS2Var)*sign(logS2SeqOffset),
      col=2)
points(log(coef(sampIGmleIG)[2]),0,pch=3)
par(oldpar)

## make a parametric boostrap to check the distribution of the deviance
nbReplicate <- 1000 #10000
sampleSize <- 100
system.time(
devianceIG100 <- lapply(1:nbReplicate,
                        function(idx) {
                          if ((idx
                          sampIG <- rinvgauss(sampleSize,mu=mu.true,sigma2=sigma2.true)
                          sampIGmleIG <- invgaussMLE(sampIG)
                          Deviance <- -2*sampIGmleIG$r(mu.true,sigma2.true)
                          logPara <- log(coef(sampIGmleIG))
                          logParaSE <- sampIGmleIG$se/coef(sampIGmleIG)
                          intervalMu <- function(n) c(-n,n)*logParaSE[1]+logPara[1]
                          intervalS2 <- function(n) c(-n,n)*logParaSE[2]+logPara[2]
                          logMuProfFct <- function(logMu,...) {
                            optimise(function(x)
                          sampIGmleIG$mll(c(logMu,x))+logLik(sampIGmleIG),...)$objective
                          }
                          logMuProfCI <- function(logMu,
                                                  CI,
                                                  a=intervalS2(4)[1],
                                                  b=intervalS2(4)[2])
                            logMuProfFct(logMu,c(a,b)) - qchisq(CI,1)/2

                          logS2ProfFct <- function(logS2,...) {
                            optimise(function(x)
                          sampIGmleIG$mll(c(x,logS2))+logLik(sampIGmleIG),...)$objective
                          }
                          logS2ProfCI <- function(logS2, CI,
                                                  a=intervalMu(4)[1],
                                                  b=intervalMu(4)[2])
                            logS2ProfFct(logS2,c(a,b)) - qchisq(CI,1)/2

                          factor <- 4
                          while((logMuProfCI(intervalMu(factor)[2],0.99) *
                                 logMuProfCI(logPara[1],0.99) >= 0) ||
                                (logMuProfCI(intervalMu(factor)[1],0.99) *
                                 logMuProfCI(logPara[1],0.99) >= 0)
                                ) factor <- factor+1
                          ##browser()
                          logMuCI95 <- c(uniroot(logMuProfCI,
                                           interval=c(intervalMu(factor)[1],logPara[1]),
                                                 CI=0.95)$root,
                                         uniroot(logMuProfCI,
                                           interval=c(logPara[1],intervalMu(factor)[2]),
                                                 CI=0.95)$root
                                         )
                          logMuCI99 <- c(uniroot(logMuProfCI,
                                           interval=c(intervalMu(factor)[1],logPara[1]),
                                                 CI=0.99)$root,
                                         uniroot(logMuProfCI,
                                           interval=c(logPara[1],intervalMu(factor)[2]),
                                                 CI=0.99)$root
                                         )
                          factor <- 4
                          while((logS2ProfCI(intervalS2(factor)[2],0.99) *
                                 logS2ProfCI(logPara[2],0.99) >= 0) ||
                                (logS2ProfCI(intervalS2(factor)[1],0.99) *
                                 logS2ProfCI(logPara[2],0.99) >= 0)
                                ) factor <- factor+1
                          logS2CI95 <- c(uniroot(logS2ProfCI,
                                           interval=c(intervalS2(factor)[1],logPara[2]),
                                                 CI=0.95)$root,
                                         uniroot(logS2ProfCI,
                                           interval=c(logPara[2],intervalS2(factor)[2]),
                                                 CI=0.95)$root
                                         )
                          logS2CI99 <- c(uniroot(logS2ProfCI,
                                           interval=c(intervalS2(factor)[1],logPara[2]),
                                                 CI=0.99)$root,
                                         uniroot(logS2ProfCI,
                                           interval=c(logPara[2],intervalS2(factor)[2]),
                                                 CI=0.99)$root
                                         )
                          list(deviance=Deviance,
                               logMuCI95=logMuCI95,
                              logMuNorm95=qnorm(c(0.025,0.975),logPara[1],logParaSE[1]),
                               logMuCI99=logMuCI99,
                              logMuNorm99=qnorm(c(0.005,0.995),logPara[1],logParaSE[1]),
                               logS2CI95=logS2CI95,
                              logS2Norm95=qnorm(c(0.025,0.975),logPara[2],logParaSE[2]),
                               logS2CI99=logS2CI99,
                              logS2Norm99=qnorm(c(0.005,0.995),logPara[2],logParaSE[2]))
                        }
                        )
            )[3]
## Find out how many times the true parameters was within the computed CIs
nLogMuCI95 <- sum(sapply(devianceIG100,
                         function(l) l$logMuCI95[1] <= log(mu.true)  &&
                         log(mu.true)<= l$logMuCI95[2]
                         )
                  )
nLogMuNorm95 <- sum(sapply(devianceIG100,
                           function(l) l$logMuNorm95[1] <= log(mu.true)  &&
                           log(mu.true)<= l$logMuNorm95[2]
                           )
```

```
                    )
nLogMuCI99 <- sum(sapply(devianceIG100,
                         function(l) l$logMuCI99[1] <= log(mu.true)  &&
                         log(mu.true)<= l$logMuCI99[2]
                         )
                  )
nLogMuNorm99 <- sum(sapply(devianceIG100,
                           function(l) l$logMuNorm99[1] <= log(mu.true)  &&
                           log(mu.true)<= l$logMuNorm99[2]
                           )
                    )
## Check if these counts are compatible with the nominal CIs
c(prof95Mu=nLogMuCI95,norm95Mu=nLogMuNorm95)
qbinom(c(0.005,0.995),nbReplicate,0.95)
c(prof95Mu=nLogMuCI99,norm95Mu=nLogMuNorm99)
qbinom(c(0.005,0.995),nbReplicate,0.99)

nLogS2CI95 <- sum(sapply(devianceIG100,
                         function(l) l$logS2CI95[1] <= log(sigma2.true)  &&
                         log(sigma2.true)<= l$logS2CI95[2]
                         )
                  )
nLogS2Norm95 <- sum(sapply(devianceIG100,
                           function(l) l$logS2Norm95[1] <= log(sigma2.true)  &&
                           log(sigma2.true)<= l$logS2Norm95[2]
                           )
                    )
nLogS2CI99 <- sum(sapply(devianceIG100,
                         function(l) l$logS2CI99[1] <= log(sigma2.true)  &&
                         log(sigma2.true)<= l$logS2CI99[2]
                         )
                  )
nLogS2Norm99 <- sum(sapply(devianceIG100,
                           function(l) l$logS2Norm99[1] <= log(sigma2.true)  &&
                           log(sigma2.true)<= l$logS2Norm99[2]
                           )
                    )
## Check if these counts are compatible with the nominal CIs
c(prof95S2=nLogS2CI95,norm95S2=nLogS2Norm95)
qbinom(c(0.005,0.995),nbReplicate,0.95)
c(prof95S2=nLogS2CI99,norm95S2=nLogS2Norm99)
qbinom(c(0.005,0.995),nbReplicate,0.99)


## Get 95 and 99% confidence intervals for the QQ plot
ci <- sapply(1:nbReplicate,
                 function(idx) qchisq(qbeta(c(0.005,0.025,0.975,0.995),
                                            idx,
                                            nbReplicate-idx+1),
                                      df=2)
             )
## make QQ plot
X <- qchisq(ppoints(nbReplicate),df=2)
```

```
Y <- sort(sapply(devianceIG100,function(l) l$deviance))
X11()
plot(X,Y,type="n",
     xlab=expression(paste(chi[2]^2," quantiles")),
     ylab="MC quantiles",
     main="Deviance with true parameters after ML fit of IG data",
     sub=paste("sample size:", sampleSize,"MC replicates:", nbReplicate)
     )
abline(a=0,b=1)
lines(X,ci[1,],lty=2)
lines(X,ci[2,],lty=2)
lines(X,ci[3,],lty=2)
lines(X,ci[4,],lty=2)
lines(X,Y,col=2)

## End(Not run)
```

---

`llogisMLE` *Maximum Likelihood Parameter Estimation of a Log Logistic Model with Possibly Censored Data*

---

## Description

Estimate log logistic model parameters by the maximum likelihood method using possibly censored data. The corresponding code for this function as well as the manual information included here is attributed to Christophe Pouzat's STAR Package (archived 2022-05-23).

## Usage

```
llogisMLE(yi, ni = numeric(length(yi)) + 1,
          si = numeric(length(yi)) + 1)
```

## Arguments

| | |
|---|---|
| `yi` | vector of (possibly binned) observations or a `spikeTrain` object. |
| `ni` | vector of counts for each value of `yi`; default: `numeric(length(yi))+1`. |
| `si` | vector of counts of *uncensored* observations for each value of `yi`; default: `numeric(length(yi))+1`. |

## Details

The MLE for the log logistic is not available in closed formed and is therefore obtained numerically obtained by calling `optim` with the `BFGS` method.

In order to ensure good behavior of the numerical optimization routines, optimization is performed on the log of parameter `scale`.

Standard errors are obtained from the inverse of the observed information matrix at the MLE. They are transformed to go from the log scale used by the optimization routine to the requested parameterization.

## Value

A list of class `durationFit` with the following components:

| | |
|---|---|
| `estimate` | the estimated parameters, a named vector. |
| `se` | the standard errors, a named vector. |
| `logLik` | the log likelihood at maximum. |
| `r` | a function returning the log of the relative likelihood function. |
| `mll` | a function returning the opposite of the log likelihood function using the log of parameter `sdlog`. |
| `call` | the matched call. |

## Note

The returned standard errors (component `se`) are valid in the asymptotic limit. You should plot contours using function `r` in the returned list and check that the contours are reasonably close to ellipses.

## Author(s)

Christophe Pouzat <`christophe.pouzat@gmail.com`>

## References

Lindsey, J.K. (2004) *Introduction to Applied Statistics: A Modelling Approach*. OUP.

Lindsey, J.K. (2004) *The Statistical Analysis of Stochastic Processes in Time*. CUP.

## See Also

`dllogis`, `invgaussMLE`, `gammaMLE`.

## Examples

```
## Not run:
## Simulate sample of size 100 from a log logisitic
## distribution
set.seed(1102006,"Mersenne-Twister")
sampleSize <- 100
location.true <- -2.7
scale.true <- 0.025
sampLL <- rllogis(sampleSize,location=location.true,scale=scale.true)
sampLLmleLL <- llogisMLE(sampLL)
rbind(est = sampLLmleLL$estimate,se = sampLLmleLL$se,true = c(location.true,scale.true))

## Estimate the log relative likelihood on a grid to plot contours
Loc <- seq(sampLLmleLL$estimate[1]-4*sampLLmleLL$se[1],
               sampLLmleLL$estimate[1]+4*sampLLmleLL$se[1],
               sampLLmleLL$se[1]/10)
Scale <- seq(sampLLmleLL$estimate[2]-4*sampLLmleLL$se[2],
             sampLLmleLL$estimate[2]+4*sampLLmleLL$se[2],
             sampLLmleLL$se[2]/10)
sampLLmleLLcontour <- sapply(Loc, function(m) sapply(Scale, function(s) sampLLmleLL$r(m,s)))
## plot contours using a linear scale for the parameters
## draw four contours corresponding to the following likelihood ratios:
##  0.5, 0.1, Chi2 with 2 df and p values of 0.95 and 0.99
X11(width=12,height=6)
layout(matrix(1:2,ncol=2))
contour(Loc,Scale,t(sampLLmleLLcontour),
        levels=c(log(c(0.5,0.1)),-0.5*qchisq(c(0.95,0.99),df=2)),
        labels=c("log(0.5)",
          "log(0.1)",
          "-1/2*P(Chi2=0.95)",
          "-1/2*P(Chi2=0.99)"),
        xlab="Location",ylab="Scale",
        main="Log Relative Likelihood Contours"
        )
points(sampLLmleLL$estimate[1],sampLLmleLL$estimate[2],pch=3)
points(location.true,scale.true,pch=16,col=2)
## The contours are not really symmetrical about the MLE we can try to
## replot them using a log scale for the parameters to see if that improves
## the situation
contour(Loc,log(Scale),t(sampLLmleLLcontour),
        levels=c(log(c(0.5,0.1)),-0.5*qchisq(c(0.95,0.99),df=2)),
        labels="",
        xlab="log(Location)",ylab="log(Scale)",
        main="Log Relative Likelihood Contours",
        sub="log scale for parameter: scale")
points(sampLLmleLL$estimate[1],log(sampLLmleLL$estimate[2]),pch=3)
points(location.true,log(scale.true),pch=16,col=2)

## make a parametric boostrap to check the distribution of the deviance
nbReplicate <- 10000
sampleSize <- 100
system.time(
            devianceLL100 <- replicate(nbReplicate,{
              sampLL <- rllogis(sampleSize,location=location.true,scale=scale.true)
              sampLLmleLL <- llogisMLE(sampLL)
              -2*sampLLmleLL$r(location.true,scale.true)
            }
                                       )
            )[3]

## Get 95 and 99
ci <- sapply(1:nbReplicate,
                 function(idx) qchisq(qbeta(c(0.005,0.025,0.975,0.995),
                                            idx,
                                            nbReplicate-idx+1),
                                      df=2)
             )
## make QQ plot
X <- qchisq(ppoints(nbReplicate),df=2)
Y <- sort(devianceLL100)
X11()
```

```
plot(X,Y,type="n",
     xlab=expression(paste(chi[2]^2," quantiles")),
     ylab="MC quantiles",
     main="Deviance with true parameters after ML fit of log logistic data",
     sub=paste("sample size:", sampleSize,"MC replicates:", nbReplicate)
     )
abline(a=0,b=1)
lines(X,ci[1,],lty=2)
lines(X,ci[2,],lty=2)
lines(X,ci[3,],lty=2)
lines(X,ci[4,],lty=2)
lines(X,Y,col=2)

## End(Not run)
```

# Index